ELSEVIER

Theoretical Computer Science 231 (2000) 129–139

Theoretical Computer Science

www.elsevier.com/locate/tcs

Denis Maurel

LI/E3i-University of Tours, 64 Avenue Jean-Portalis, F37200 Tours, France

Abstract

The algorithm that we present here builds an acyclic deterministic finite state machine (automaton or transducer), as each word recognized has a proper element, i.e. a transition or a final state that belongs only to the recognizing path of this word. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Finite state automaton; Transducer; ADFA; Electronic dictionary

1. Motivation

Automata theory is a convenient tool for the representation of linguistic phenomena [1, 9]. Our purpose is to build an acyclic deterministic finite state automaton (ADFA), or transducer, as each word recognized has a *proper element*, i.e. a transition or a final state that belongs only to the recognizing path of this word (Section 4).

It is particularly interesting to use a pseudo-minimal transducer when one simultaneously want to read and to modify a transducer. For example, we use it to represent a list of words with their number of tokens: The number of tokens of a reading word has to be indented in the same time. Of course, we can also build a pseudo-minimal machine before an algorithm to minimize it (see [10]), because it is minimized in part and, therefore, size cheap.

Our algorithm is based on the automata pseudo-minimization algorithm [8] and the minimization of finite-state transducer algorithm [6]. First, we describe it with examples (Section 2), then we precisely define the result we want to obtain (Section 3). We have named it a *pseudo-minimal automaton* (Definition 3) and a *pseudo-minimal transducer* (Definition 4). Finally, we present the algorithm to build a pseudo-minimal machine (Section 5) and we give some computational results (Section 6).

2. Examples

In this section, we are explaining our algorithm with some examples. Assume that our alphabet consists of words used in date adverbials [3].

E-mail address: maurel@univ-tours.fr (D. Maurel)

PII: S0304-3975(99)00022-5

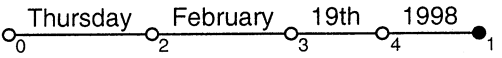

Fig. 1. The pseudo-minimal automaton: first step.

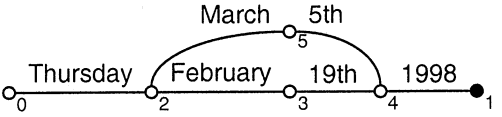

Fig. 2. The pseudo-minimal automaton: second step.

2.1. Prefixes and suffixes

First, we want to build an automaton to recognize these three dates:

Thursday February 19th 1998
Thursday March 5th 1998
Thursday March 19th 1998

We initialize our automaton with two states, 0 and 1, but without transition. The state 1 is said the *terminal* state. Then, we add to it the first date, like in Fig. 1.

Then we add the second date: We read the prefix on the automaton from the state 0 up to the state 2 and the suffix from the state 1 down to the state 4. And we create one state and two transitions between the states 2 and 4 to complete this date (Fig. 2).

However, we do not use the same way for the third date: its prefix leads to the state 5 and the suffix starts with the state 3! It does not make sense. So we need a definition of the couple (prefix, suffix).

Definition 1. Let an alphabet L and w a word of $L^\star$. We say that $(p,s) \in L^\star \times L^\star$ is a couple (prefix, suffix) of w if, and only if:

$$p = w \quad \text{and} \quad s = \varepsilon \text{ (the empty word)}$$

or

$$\exists b \in L^\star,\ b \neq \varepsilon \quad \text{and} \quad m = pbs.$$

For instance, the word May has five couples (prefix, suffix):

$$(\text{May}, \varepsilon), (\text{Ma}, \varepsilon), (\text{M}, \varepsilon), (\varepsilon, \varepsilon), (\text{M}, y).$$

Now, we can say that the prefix of the third date leads to the state 5 and the suffix starts with the state 4. And we create one transition between the states 5 and 4 to complete this date (Fig. 3).

2.2. Convergent state encounter

When we read the prefix of a word to add at the automaton, if we lead to a state with more than one in-transition (a *convergent* state), we have to duplicate this state, at the risk of recognizing a word out of our list.

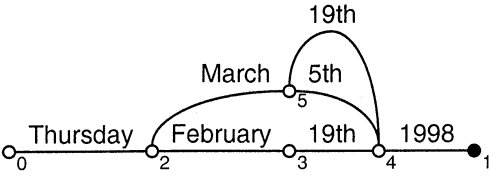

Fig. 3. The pseudo-minimal automaton: third step.

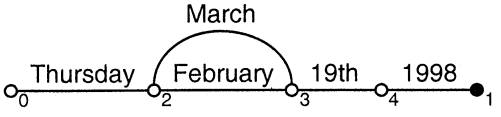

Fig. 4. The pseudo-minimal automaton: first and second steps.

For instance, see the following list of three dates, that is the same list than before, but in an other order:

Thursday February 19th 1998
Thursday March 19th 1998
Thursday March 5th 1998

Fig. 4 shows the two first steps of the automaton building.

If we say that the prefix of the third date leads to the state 3 and that its suffix starts with the state 4: and if we create one transition between the states 3 and 4: we make a mistake, because we recognize also the date *Thursday February 5th 1998* that is not in our list.

So, we have to duplicate this state in a state 5 and we say that the prefix leads to this state. Now, we can create one transition between the states 5 and 4. And we find Fig. 3 again.

2.3. Minimal and pseudo-minimal automaton

Until now, all these pseudo-minimal automata are also minimal. That is not true for the following example: We are adding to our list, the date *Thursday February 5th 1998*. The prefix of this fourth date leads to the state 3 and the suffix starts with the state 4 (Fig. 3). So, we have just to create one transition between the states 3 and 4 to complete this date (Fig. 5).

This automaton is not minimal. See the nimimal automaton to recognize this list in Fig. 6.

2.4. Divergent state encounter

When we read the suffix of a word to add at the automaton, if we lead to a state with more than one out-transition [1] (a *divergent* state), we need to duplicate this state, always for the same reason, to recognize a word out of our list.

[1] Or a final state.

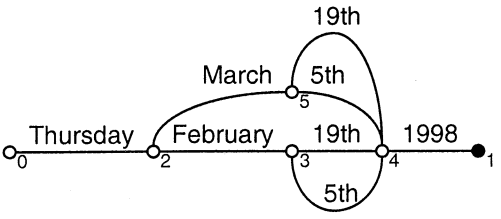

Fig. 5. The pseudo-minimal automaton: fourth step.

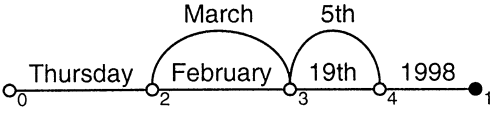

Fig. 6. The minimal automaton.

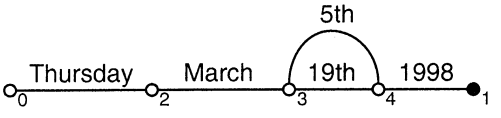

Fig. 7. The pseudo-minimal automaton: first and second steps.

For instance, see the following list, that is yet the same list than before, but in a third order:

Thursday March 19th 1998
Thursday March 5th 1998
Thursday February 19th 1998

Fig. 7 shows the two first steps of the automaton building.

We cannot say that the prefix of the third date leads to the state 2 and that its suffix starts with the state 3: because, if we create one transition between the states 2 and 3, we recognize also the date *Thursday February 5th 1998*. So, we duplicate the state 3 in a state 5 and we create one transition between these two states. We find a third time in Fig. 3 (with a renaming of the states 3 and 5).

2.5. Unambiguous transducer

We build the underlying automaton as before and we put the output as left as possible [6]. First, we put it before the state 0, in the initial output [7]. And later, we move the uncommon suffixes left to right.

For instance, we want now to associate to a date, its day in the week, as the following:

February 19th 1998 → *Thursday*
February 17th 1998 → *Tuesday*

Fig. 8 shows the first step of the transducer building, and Fig. 9, the second step.

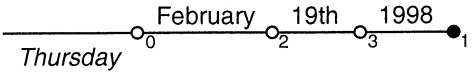

Fig. 8. The pseudo-minimal transducer: first step.

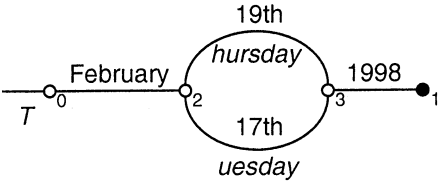

Fig. 9. The pseudo-minimal transducer: second step.

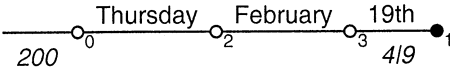

Fig. 10. The pseudo-minimal transducer: first and second steps.

2.6. Ambiguous transducer

It is posible that a list has homonyms, i.e. the same input, but different outputs. Then, the transducer that recognizes it, is ambiguous [9]. Since we do not want that a pseudo-minimal transducer is ambiguous, we add to the set of outputs a disjunctive symbol (|) to make it unambiguous.

For instance, we want now to associate to a date, its day in the week, as the following:

Thursday February 19th $\rightarrow$ *2004*
Thursday February 19th $\rightarrow$ *2009*

As shown in Fig. 10, the uncommon suffixes of these two outputs become only one output: 4|9. We can notice that a disjunctive output is always on a final state or on an in-transition of the terminal state (a *terminal* transition), because we do not know that we read an homonym of an other line before the last letter. The state 3 has a disjunctive out-transition, we will say that it is *divergent*.

3. Definitions

After these examples, we have to give exact definitions. It is the purpose of this section.

3.1. Notations

In our definitions, we use some notations:

- Q is a non-empty finite set of states
- L is a non-empty finite set of letters (input alphabet)

- Ω is a non-empty finite set of letters (output alphabet)
- q_0 is the initial state
- σ_0 is the initial output
- F is a non-empty subset of Q (final states)
- δ is a partial function: $Q \times L \to Q$ (input transition function)
- λ is a partial function: $Q \times L \to \Omega^\star \cup \{|\}$ (output transition function)
- σ is a partial function: $F \to \Omega^\star \cup \{|\}$ (output final state function)
- ε is the empty word
- $\underline{\delta}$ is the extended input transition function: $Q \times L^\star \to Q$ [2]
- $\underline{\lambda}$ is the extended output transition function: $Q \times L^\star \to \Omega^\star \cup \{|\}$
- $\bigwedge(w_1, w_2, \ldots, w_n)$ the common prefix of words $w_1, w_2, \ldots, w_n \in \Omega^\star \cup \{|\}$ [2]

And we make some states and transitions different:

Definition 2. Four points:

1. A terminal state (Section 2.1) q is a final state with no out-transition:

$$q \in F \quad \text{and} \quad \forall l \in L,\ \delta(q,l) \text{ is not defined.}$$

2. A terminal transition (Section 2.6) is an in-transition of a terminal state:

$$q \in Q \quad \text{and} \quad l \in L \quad \text{and} \quad \delta(q,l) \text{ is terminal.}$$

3. A divergent state q is a final (but non terminal) state, a state with more than one out-transition (Section 2.4) or a state with a disjunctive out-transition (Section 2.6).

$$q \in F \quad \text{and} \quad \exists l \in L,\ \delta(q,l) \text{ is defined.}$$

or

$$q \in Q \quad \text{and} \quad \exists l,\ l' \in L,\ l \neq l' \quad \text{and} \quad \delta(q,l),\ \delta(q,l'), \text{ are defined.}$$

or

$$q \in Q \quad \text{and} \quad \exists l \in L,\ | \text{ is a letter of the word } \lambda(q,l).$$

4. A convergent state q (Section 2.2) is a non-terminal state with more than one in-transition:

$$q \in Q \quad \text{and} \quad \exists l', l'' \in L,\ \exists q', q'' \in Q,$$

$$(q', l') \neq (q'', l'') \quad \text{and} \quad \delta(q', l') = \delta(q'', l'') = q.$$

We denote by Δ, the subet of Q consisting of the divergent states, and Γ, the subset of Q consisting of the convergent states.

[2] We assume that $\bigwedge(w_1|w_2) = \bigwedge(w_1, w_2)$.

3.2. Pseudo-minimal machines

Now, we can define successively a pseudo-minimal automaton and a pseudo-minimal transducer:

Definition 3. A pseudo-minimal automaton is an acyclic deterministic finite state automaton (ADFA), with one, and only one, terminal state q_1, and that satisfies the following conditions:

1. if q is a divergent state, there is one and only one path from q_0 to q:

$$\forall q \in \Delta,\ \exists! w \in L^\star,\ \underline{\delta}(q_0, w) = q;$$

2. if a convergent state is reached by two transitions with the same label, one (at least) of the source states of these transitions is divergent:

$$\forall q \in \Gamma,\ \exists q', q'' \in Q,\ \exists l \in L,$$

$$q' \neq q'' \quad \text{and} \quad \delta(q', l) = \delta(q'', l) = q \Rightarrow q' \in \Delta \text{ or } q'' \in \Delta.$$

Automata obtained in Section 2 are indeed acyclic, deterministic (by amalgamating states and transitions taking part in a common prefix) and also minimized in part (by amalgamating states and transitions taking part in a common suffix). When we read the common prefix of the new word and the automaton that is being constructed, if we arrive at a convergent state q (Section 2.2), we duplicate this state; hence, any divergent state cannot be reached in more than one way (Definition 3.1). When we read the common suffix of the new word and the automaton, if we arrive at a divergent state (Section 2.4), we duplicate this state; so, it is impossible to reach a state with two transitions with the same letter, except a convergent state (Definition 3.2).

Note that the pseudo-minimal automaton obtained is unique up to a renaming of the states i.e. the result is independent of the order of reading the input strings on the list (the original algorithm of Revuz does not have this property).

Definition 4. A pseudo-minimal transducer is an unambiguous subsequential finite state transducer that satisfies the following conditions:

1. The underlying finite state automaton is a pseudo-minimal automaton
2. The common prefix of all outputs given after a state is empty:

$$\forall q \in Q \backslash F,\ \forall l \in L,\ \bigwedge_{\{\delta(q,l)\ \text{is defined}\}} (\lambda(q, l)) = \varepsilon,$$

$$\forall q \in F,\ \forall l \in L,\ \bigwedge \left(\bigwedge_{\{\delta(q,l)\ \text{is defined}\}} (\lambda(q, l)), \sigma(q) \right) = \varepsilon.$$

The output of a recognized word w is $\underline{\lambda}(q_0, w)\sigma(\underline{\delta}(q_0, w))$. We can notice a corollary of Definition 4.2 [3]:

[3] Because $\bigwedge(w) = w$.

Corollary 1. $\lambda(q,l)\neq\varepsilon\Rightarrow q\in\Delta$.

To obtain the pseudo-minimal transducer, we must distribute the output strings as close as possible to q_0 (Section 2.5). Thus, during the pseudo-minimization, it is necessary to compare the output strings while searching the longest common prefix, to keep the common output prefix, and to merge (by concatenation) the output suffix with output strings of the following transitions, or to distribute the output suffix on a final state. That is exactly the meaning of Definition 4.2.

4. The proper element

Two different words in a list give two different paths in a pseudo-minimal automaton, and, thus, a divergent state. Because Definition 3.1, we cannot reach a divergent state by more than one path. So, for each recognized word, there exists at least one transition defined for this word only (a *proper* transition) or a final state which is final for this word only (a *proper* state). We are giving a new definition:

Definition 5. Let w be a word recognized by an automaton:
1. if there exists, the first proper transition on the path of w is said to be the proper element of w;
2. else, if the last final state on the path of w is final for this word only, we name it a proper element of w.

If the proper element of a recognized word w exists, we denote it by $\Pi(w)$. We just have seen that each recognized word, by a pseudo-minimal automaton, has a proper element. We note the following corollary of Definitions 2.3 and 5:

Corollary 2. $\forall q\in Q,\ \exists w\in L^{\star},\ \Pi(w)=q$ *or* $\Pi(w)=(q,l,q')\Rightarrow q\in\Delta$.

For instance, in Fig. 5, the proper elements are:

$$\Pi(Thursday\ February\ 19th\ 1998)=(3,\ 19th,\ 4)$$

$$\Pi(Thursday\ March\ 5th\ 1998)=(5,\ 5th,\ 4)$$

$$\Pi(Thursday\ March\ 19th\ 1998)=(5,\ 19th,\ 4)$$

$$\Pi(Thursday\ February\ 5th\ 1998)=(3,\ 5th,\ 4)$$

And states 3 and 5 are divergent. But, in the minimal automaton of Fig. 6, no word has a proper element: Generally, a minimal automaton does not associate a proper element to a given word.

5. The algorithm

Table 1 presents the skeleton of the algorithm: We initialize our automaton with two states, q_0 and q_1 (Section 2.1), then we read the first line and add the first word to the automaton and we put the initial output before state q_0 (Section 2.5). Finally, we

Table 1
The algorithm

```
Creation of initial state q0 and terminal state q1
Reading the first line
Creation of states and transitions between q0 and q1
The first output → σ0
WHILE the end of the list is not reached
   Handling of the prefix
   Handling of the suffix
   Middle of the word
ENDWHILE
```

Table 2
Handling of the prefix

```
Read a word w and its output string ω
q0 → prefix
0 → p
length(w) − 1 → s
WHILE w is not finished
AND (prefix, w[p], q) is read on the transducer
AND q ≠ q1
   IF q is a convergent state THEN duplicates q ENDIF
   handling of the common prefix of output
   q → prefix
   p + 1 → p
ENDWHILE
If (w is finished)
THEN
   F ∪ {prefix} → F
   put ambiguities on σ(prefix)
   CONTINUE
ENDIF
IF (q = q1)
THEN
   IF the word is finished
   THEN
   put ambiguities on λ(prefix)
   CONTINUE
ENDIF
duplicates q
handling of the common prefix of output
q → prefix
p + 1 → p
ENDIF
```

Table 3
Handling of the suffix

$q_1 \rightarrow suffix$
WHILE $s > p + 1$
AND $(q, w[s], suffix)$ is read on the transducer
AND q is not a divergent state
AND $\lambda(q, w[s]) = \varepsilon$
  $q \rightarrow suffix$
  $s - 1 \rightarrow s$
ENDWHILE

Table 4
Example files

Files	Lines	Codes	Codes/lines (%)	KBits
1	3117	94	3	50
2	38 740	38 740	100	800
3	50 149	6	0.01	677
4	99 1670	120	0.0001	26 348

read the whole list, searching the prefix and the suffix of each word, and adding new states or transitions to recognize the middle of the word.

When we search the prefix (Table 2), we read letter by letter and we move forward one state (if necessary, we duplicate the state – Section 2.2), except at the end of the prefix or the word, and if we reach q_1. We note the length of the reading word for the handling of the suffix.

When we search the suffix (Table 3), we read back letter by letter and we move forward one state, except at the end of the suffix, and if we reach a divergent state (Section 2.4) or a transition with an output. We duplicate later, when we create states and transitions for the middle of the word.

6. Implementation

We implement this algorithm in C language and we present here four different examples (Table 4). The first third lists have been built in the *Prolex* project on proper names [4, 5]:

1. A list with some codes: 343 cities associated to the number of its department (French territorial division).
2. A list with one different code for each French city (a key of a database, represented by one digit, a period and five digits).
3. A list of French place name, with a lot of homonyms and with very few codes.

And the fourth list is a dictionary of almost one million words.

Table 5 presents the size of the resulting pseudo-minimal transducers. Except if there is a good factorization of the codes, outputs are more numerous than codes that are

Table 5
The pseudo-minimal transducers

Files	States	Transitions	Outputs
1	9015	11 991	108
2	66 379	98 827	32 726
3	120 498	157 194	46
4	239 594	311 539	119

concatenation of outputs. The great difference between the number of codes of the third list is understandable by a lot of disjunctive codes.

Acknowledgements

The author thanks Mehryar Mohri for a lot of discussions and works that are at the origin of this paper.

References

[1] M. Gross, D. Perrin, Electronic Dictionnaries and Automata in Computational Linguistics, Lecture Notes in Computer Science, vol. 377, Springer, Berlin, 1989.
[2] Hopcroft Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[3] D. Maurel, Building automaton on Schemata and Acceptability Tables, Lecture Notes in Computer Science, vol. 1260, Springer, Berlin, 1997, pp. 72–86.
[4] D. Maurel, C. Belleil, E. Eggert, O. Piton, Le projet PROLEX, séminaire Représentations et Outils pour les Bases Lexicales, Morphologie Robuste de l'action Lexique du GDR-PRC CHM, Grenoble, 1996.
[5] D. Maurel, C. Belleil, O. Piton, E. Eggert, Un dictionnaire électronique relationnel pour les noms propres, Revue française de Linguistique appliquée, 2-1 (1997) 101–111.
[6] M. Mohri, Minimization of Sequential Transducers, Lecture Notes in Computer Science, Springer, Berlin, 1994a, p. 807.
[7] M. Mohri, Finite-state transducers in language and speech processing, Comput. Linguistics 23-2 (1997) 269–311.
[8] D. Revuz, Dictionnaries et lexiques – Méthodes et algorithmes, Doctoral dissertation in Computer Science, University of Paris, VII, 1991.
[9] E. Roche, Y. Schabes (Ed.), Finite State Language Processing, MIT Press, Cambridge, MA, 1997.
[10] B.W. Watson, Taxonomies and Toolkits of Regular Language Algorithms, Thesis, Technische Universiteit Eindhoven, 1995.